Chapter 11: Configuring and Launching CRL

This chapter describes the configuration tasks that are necessary for running the application, for customizing the desktop environment to your experiment, and for enabling particular features of **CRL**. It also discusses invoking the application.

11.1 Editing the Properties File

11.1.1 The Bare Minimum

Before you can even invoke an installed instance of **CRL**, you must set a crucial subset of the parameters in the properties file. The properties file, `LogbookConfigParms.properties`, is stored in the `LogBook_admin` subdirectory[1].

In this section we just list the properties you need to edit. Look up each property as needed in Chapter 15: *CRL's Java Properties* to get information about setting its value.

Before editing the properties file, you must have information about your directory structure and your database. You're welcome to set all the parameters at this point, but here's the bare minimum that MUST be set:

```
Logbook.file_location.entry_directory
Logbook.database.enabled
Logbook.database.vendor
Logbook.database.version
Logbook.database.driver
```

If the connection URL (location of the database) is not of the form `<protocol>//<machine_name>/<database>?user=<username>&password=<password>`, then set the parameter:

```
Logbook.database.connection_url
```

1. In the **CRL** 1_7_04 release for Windows, the default **CRL** top level directory on Windows is `CRL\CRLadmin\config`. The `LogBook_admin` subdirectory should be under this.

If you set the above parameter, ignore the following five. Else, ignore above parameter and instead set the following parameters:

```
Logbook.database.protocol
Logbook.database.server
Logbook.database.dbms_name
Logbook.database.username
Logbook.database.password
```

Once you've made the initial edits to the properties file, launch the **CRL** application (see section 11.3 *Launching the CRL Application*) to see what it looks like and how it works. It comes with a sample configuration. Looking it over and trying things out will help you visualize and plan the desktop configuration for your experiment. Then go to section 11.4 *Configuring Your Desktop*.

11.1.2 Further Edits

After you become somewhat familiar with the desktop and operation of **CRL**, come back to the properties file and set additional parameters as you are ready to enable or change various features. Refer to Chapter 15: *CRL's Java Properties* for help on setting values for these parameters.

11.2 Configuring CRL to Use a Preinstalled JVM (Optional)

The JVM is included as part of the **CRL** release. We have extended the functionality of the JVM we ship in order to enable the IPen® functionality in the **CRL**. You can use another installation of the JVM (either JDK or JRE[1], v1.4.1 or higher), but to have IPen® functionality, you need to copy several files from the shipped JVM to the one you plan to use. You also need to modify the **CRL** invocation script so that it runs your JVM.

11.2.1 Files to Copy

All the files you'll need to copy from the shipped JVM reside under `$CRL_DIR/jdk<x.y>/jre/`, where `<x.y>` refers to the version. First **`cd`** to that directory:

1. The JDK stands for Java Development Kit. The JRE stands for Java Runtime Environment. Both include a Java Virtual Machine (JVM) JVM. The JDK is a superset of the JRE, and adds files that the developer needs in order to develop Java programs, not just run them.

```
% cd $CRL_DIR/jdk<x.y>/jre/
```

Now start the copies:

1) Copy the files `comm.jar` and `jcl.jar` from `$CRL_DIR/jdk<x.y>/jre/lib/ext/` to the `jre/lib/ext/` directory under your JVM, e.g.,:

```
% cp lib/ext/comm.jar \
  /path/to/your/java/jdk<x.y>/jre/lib/ext/

% cp lib/ext/jcl.jar \
  /path/to/your/java/jdk<x.y>/jre/lib/ext/
```

2) Copy the file `javax.comm.properties` from `$CRL_DIR/jdk<x.y>/jre/lib/` to the `jre/lib/` directory under your JVM, e.g.,:

```
% cp lib/javax.comm.properties \
  /path/to/your/java/jdk<x.y>/jre/lib/
```

3) Copy the files `libSerial.so` and `libParall.so` from `$CRL_DIR/jdk<x.y>/jre/lib/i386` to the `jre/lib/i386` directory under your JVM and change mode to 777, e.g.,:

```
% cp lib/i386/libSerial.so \
  /path/to/your/java/jdk<x.y>/jre/lib/i386

% cp lib/i386/libParall.so \
  /path/to/your/java/jdk<x.y>/jre/lib/i386

% cd /path/to/your/java/jdk<x.y>/jre/lib/i386

% chmod 777 libSerial.so

% chmod 777 libParall.so
```

11.2.2 Sample CRL Invocation Script

A simplified script for **CRL** is listed here (the last line has been abbreviated, and is described below; `<x.y>` is used in place of the JRE version.) It is included for your information:

```
#!/bin/bash

INSTALLDIR=$CRL_DIR
VERSION=V<x_y>
JARLIB=$INSTALLDIR/LogBook$VERSION/

LOGBOOK_ADMIN_DIR=$HOME/.crl

echo Running Logbook version $VERSION in $INSTALLDIR

JARS="$JARLIB/LogBook_logentry$VERSION.jar:$JARLIB/LogBook_xmlbeans$VERSION.
jar:$JARLIB/LogBook_xmldatatypes$VERSION.jar"
MAIN_JAR=$JARLIB/LogBook$VERSION.jar
COMM_JAR=$CRL_DIR/jre<x.y>/lib/ext/comm.jar
ALL_JARS=$JARS:$MAIN_JAR:$COMM_JAR
```

```
JVM = $CRL_DIR/jre<x.y>/bin/java

$JVM <options> -classpath $ALL_JARS logbook/LogBook $LOGBOOK_ADMIN_DIR
```

The last line of the script is the command that invokes the JVM (and thereby, **CRL**). The five strings on the command line are the following items:

`$JVM` the path to the JVM installation

`<options>` various options to JVM

`-classpath $ALL_JARS`

the last option to JVM; this specifies a search path for application classes and resources, which the **CRL** developers have put into various jar files. (A java jar file is an archive of many java classes put into one file.)

`logbook/LogBook`

the java class to execute

`$LOGBOOK_ADMIN_DIR`

a parameter which tells **CRL** the parent directory of the directory containing **CRL**'s properties file

11.2.3 Modify the CRL Invocation Script

The second-to-last line of the script is an environment variable definition that sets the path to the JVM that gets invoked:

```
JVM = $CRL_DIR/jre<x.y>/bin/java
```

It's easiest to simply change the `JVM` environment variable definition and leave the command line (the last line) unchanged. This will make the command run your JVM installation rather than the one shipped with **CRL**.

11.3 Launching the CRL Application

Before running **CRL**, you must perform at least a minimum of configuration and the database must be running. VERY IMPORTANTLY, YOU MUST EDIT THE PROPERTIES FILE! See section 11.1 *Editing the Properties File*. **CRL** may not even run if you neglect this step. And if it does run, it most likely will not be set up the way you want it.

11.3.1 Once you're ready for launch...

An "FYI": The command you use to launch the application actually runs a java command (usually as part of a script) which sets the java class path, optionally defines variable names for use in the properties file, and lastly furnishes the **CRL** top level directory path (as set in the installation) to the application.

11.3.2 Windows (Local and AFS Installations)

On Windows, you will have a desktop icon for **CRL**. Just double-click it to run the program.

11.3.3 Linux (Local UPD Installation)

Run **`setup`** to add the directory containing the **CRL** script to your $PATH, then run the **`crl`** command:

```
% setup crl V1_<x_y> [-f Linux]
% crl
```

11.3.4 Linux (Local Tar File Installation)

For a Linux tar file installation, manually add the directory containing the **CRL** script to your $PATH, then run the program by entering the script name:

```
% <crlscriptname>
```

11.3.5 Linux (AFS Installation)

Running the **CRL** installation in Fermilab's AFS product area assumes that the setup described in section 10.6 *Preparing to Use CRL in Fermilab's AFS Products Area* has been done. Your machine must be running AFS. There should be a script on your local machine that runs **CRL** such that it points to local configuration information.

If you are logged into an fnalu Linux node (for example, flxi02.fnal.gov), or any Linux node that has a **UPS** database setup for the AFS products area, launching the application should be as simple as:

```
% setup crl V<x_y> [-f Linux]
% <local_crl_scriptname>
```

If you are on a Linux system with no **UPS** database, you may need to set an environment variable that points to the product in AFS space, for example:

```
% setenv CRL_DIR /afs/fnal.gov/ups/crl/V<x_y>/Linux
```

and then run the local script that invokes **CRL**:

```
% <local_crl_scriptname>
```

11.4 Configuring Your Desktop

To configure the various elements of the desktop, you need to edit the `LogBookConfig.xml` configuration file. There are also XML configuration files for inquiries and forms.

Never edit the DTD file that goes with an XML file! The **CRL** application code depends on the DTD structure.

`LogBookConfig.xml` is found in the `LogBook_admin` directory. All the XML you need to know in order to understand and edit the configuration file can be found in Chapter 16: *Introduction to XML and DTD Files*. The files themselves are described in Chapter 17: *The CRL Desktop Configuration File*.

11.4.1 Define Keywords

Keywords may not be the first thing you want to configure, but we place this topic first in this section because you can attach keywords to many of the items that follow. If you read about keywords first, then at least you'll know how and where to go back and enter them.

Keywords can be configured to link to logbook entries in order to provide an additional dimension for querying the database when attempting to later identify and retrieve particular entries. Keywords are stored in UPPERCASE.

Each **input container** may have its own set of default, "attachable" keywords pertaining to the container topic. The keywords may be configured for the topic itself, or for a menu/submenu that leads to it. Any or all of these keywords may be configured to link automatically to each entry in the container. You can configure each automatically linked keyword such that it is removable, or not.

Similarly, each **data type** may have keywords associated with it. In this case, every logbook entry of a given data type inserted into a given input container would have the same set of default keywords, and users can choose from among them.

A **desktop page** may also have keywords configured for it; these keywords would be available for all containers, and for all data types on the given page. (These are only meaningful on entry input pages.)

The **CRL application as a whole** may also have keywords configured; these keywords would be available to the user for all containers, and for all data types on all pages.

See section 17.2.3 *Keyword* for configuration information.

11.4.2 Define a new Desktop Page (Data Entry or Report)

A desktop page is a work space in **CRL**. There may be several pages to your desktop; pages are configurable by experiment. Only one page is visible and active at a time.

A desktop page may be configured for data entry and manipulation, or for searching/viewing/manipulating archived entries only. The former is typically called a "data entry page" or "entry-input page", and the latter a "report page". All pages provide one or more menus for the user, and each data entry page also provides a data-entry toolbar.

- The element type `EntryInputPage` is used to define each of the desktop pages on which logbook entries can be made; see section 17.2.6 *EntryInputPage* for information on constructing the XML code.
- The element type `Page` is used to define each of the non-data-input desktop pages in the application, e.g., a page for reports only; see section 17.2.5 *Page* for information on constructing the XML code.

11.4.3 Define a Menu and/or Submenu on a Desktop Page

Each desktop page has a set of menu headings lined up horizontally underneath the page title. These are pull-down menus. These pull-down menus may cascade several levels in order to allow precise categorization of entries or reports.

Menus may be defined for both types of pages, data entry and report (see section 11.4.2 *Define a new Desktop Page (Data Entry or Report)*. On entry input pages, menus are intended to represent general logbook entry categories. You can also include report menus on an entry-input page, but not vice-versa. Report menus represent general reporting categories, e.g., daily report.

Make the menu names and options descriptive!

The menu headings and all the sublevels of categorization except the final one correspond to logbook entry *categories*. The final level of menu categorization (i.e., an option on the lowest-level submenu, or on the menu itself in the absence of submenus) is considered the *topic*. A *container* is associated with a topic.

- The element type `Menu` is used to define a top level menu on a desktop page; see section 17.2.7 *Menu and SubMenu*.

- The element type `Submenu` is used to define each of the submenus coming off a menu or a higher-level submenu; see section 17.2.7 *Menu and SubMenu*.
- The element type `Topic` is used to define each of the menu options on the lowest-level submenu, or on the menu itself in the absence of submenus; see section 17.2.8 *Topic*.

11.4.4 Define a ToolBar with ToolButtons for Data Entry Types

You must configure a data entry toolbar for each desktop page that allows logbook data entry. A toolbar must therefore appear in the declaration of each `EntryInputPage` element. It will display vertically down the right-hand side of the page.

A toolbar must include at least one toolbutton for each logbook entry type that you want to make available on the associated page (e.g., text, plain text, execute command, application output file, form(s), etc.).

The element type `ToolBar` is used to define a toolbar on an entry input desktop page and the element type `ToolButton` is used to define a button on the toolbar. The images that come with the default configuration are included in the **CRL** jar file. See section 17.2.9 *ToolBar and ToolButton* for information on constructing the XML code.

Creating New Toolbutton Images

You can create additional images for toolbuttons. Toolbuttons may be graphic images (`.gif` files) or plain text. We recommend `.gif` files because they look nicer. First make sure you've got a directory to contain these images. It must be located under the same directory that contains the `Logbook_admin` directory, and may be called anything (the default is `images/entryinputpages`). The template file `Button.gif` is provided in this default directory; edit it to make other buttons that match the default ones. Include the whole path in the XML configuration file when pointing to one of these image files.

11.5 Creating Configuration Files for Forms

CRL comes with some ready-made forms that you can use as is, modify, or delete, as you like. You can also create new forms. Each form entry type you add to your **CRL** installation has its own XML form definition file. The `form.dtd` file and the XML form definition files must be located in a

directory defined by the `Logbook.file_location.forms_directory` parameter in the Properties file (described in Chapter 15: *CRL's Java Properties*).

The XML elements allowed in a form definition file are listed in section 18.1 *Form Definition Files*, along with examples.[1] To refresh your memory on XML elements and attributes, see section 16.3 *Element Types and Attributes in the DTD File*.

11.5.1 Create/Modify the XML Form Definition Files

There are several things you need to know up front about creating new forms:

- For each new form, a corresponding data entry toolbutton must be added to the toolbar. See section 11.4.4 *Define a ToolBar with ToolButtons for Data Entry Types*.
- The XML tags are not case-sensitive.
- You can create forms with text areas, radio buttons, check boxes, selects, tables and lists. You can combine these elements on the same line, if you like.
- You can group elements on a line so that they are arranged more attractively when viewed in HTML on the web (appearance in **CRL** container is not affected). See section 18.1.7 *Sample Lines with Field Placement Grouping*.
- With the exceptions of `<Form>` and `<RepeatBlock>`, all form elements (tags) must be contained within a `<Line> ... </Line>` tag; further, a `<Line>` element must not contain either of the above-mentioned element types.
- The entire form and/or individual form elements can be aligned center (the default), right, or left. Both the `<Form>` and `<Line>` elements support the `align` attribute.
- Forms can be configured such that the entry gets automatically emailed (in HTML format) to one or more individual addresses and/or to one or more mail lists at the time it is archived (see section 11.5.2 *Enable Automatic Electronic Mailing of Form Entries*).
- You can set up a table in your form definition file by specifying the columns and rows of data. See section 11.5.3 *Include Tables in a Form*. On a form entry containing a table, two buttons are displayed for the user: **ADD NEW ROW** and **DELETE A SELECTED ROW**. These buttons are associated with the table portion of the entry, and appear above it.

1. Unlike the other XML configuration files in **CRL**, the form definition files are not strictly governed by a DTD file; however the file is referenced and must be present.

- You can have your form run a program, the output of which will appear in a text area on the form when a user creates an entry using this form.
- Forms may contain embedded forms via the element `<insertform>`. This enables you to create end forms in which some fields are reloadable and others are not. This technique is described in section 11.5.4 *Create Forms with Selected Reloadable Fields*.
- Forms may contain "repeat blocks". From the user's point of view a repeat block is a portion of the form entry that is demarcated and displayed along with a **REPEAT** button, which when clicked causes that portion of the form to be duplicated in the entry. From the administrator's point of view, a repeat block is a portion of the form enclosed between `<REPEATBLOCK> ...</REPEATBLOCK>`, intended for said purpose.

Repeat blocks may contain one or more `<LINE>...</LINE>` elements only; they cannot contain the `<insertform>` element.

11.5.2 Enable Automatic Electronic Mailing of Form Entries

Entries can be sent to email recipients automatically upon archive, or manually by the user. The latter method applies to all data entry types, and is discussed in section 4.7 *Sending Entries via Email*. Automatic mailing applies only to form entries.

In order to enable automatic electronic mailing of form entries, you need to do one of the following:

1) create a mail list file (format and location given below) and insert the filename into the configuration for the toolbutton that corresponds to the form, or

2) insert a destination email address directly into the configuration for the toolbutton that corresponds to the form. It must contain the @ symbol. Optionally, also include a "from" email address and a subject line.

See section 17.2.9 *ToolBar and ToolButton* for the XML format (there is an example of this in the sample code given there). If a mail list file exists and is specified, it will take precedence over email information contained in the **CRL** configuration file.

If you wish to create a mailing list for any of your forms, first create a text file in the directory specified by the parameter `Logbook.file_location.mail_list_directory` in the properties file (Chapter 15: *CRL's Java Properties*). The text file can have any name, but its contents must conform to the format of the following sample file and to the constraints listed below:

```
<MAILLIST>
 <TO>user1@fnal.gov</TO>
```

```
 <TO>user2@fnal.gov</TO>
 <TO>userxyz@myuniv.edu</TO>
 <TO>listxyz@myuniv.edu</TO>
 <FROM>user3@fnal.gov</FROM>
 <FROM>listabcd@fnal.gov</FROM>
 <CC>powersthatbe@fnal.gov</CC>
 <BCC>mefistofele@underworld.org</BCC>
 <SUBJECT>Muon chambers update $D $T</SUBJECT>
</MAILLIST>
```

Your mail list file may contain any number of any of these elements (`<TO>...</TO>`, `<FROM>...</FROM>`, `<CC>...</CC>`, and `<BCC>...</BCC>`) in any order. Each element can contain only one email address. An email address may be an individual address or a mailing list.

The file may also contain one subject element (`<SUBJECT>...</SUBJECT>`), in any position with respect to the other elements. The macros `$D` and `$T` may be used in the subject line; they get replaced by the current date and time, respectively, when the form is archived and the email is sent.

11.5.3 Include Tables in a Form

You can set up a table in your form definition file by specifying the columns and rows of data. The data types that can be inserted into cells of a table include:

`DateAndTime`	current date and/or time
`CheckBox`	boolean
`Integer`	whole number
`Double`	floating point number
`Select`	pull-down, editable or noneditable selection box
`Field`	text

The syntax for setting up a table within a form is as follows (where `datatype_<n>` refers to one of the above data types):

```
<Line>
  <Table>
    <ColumnLabel name="title of column1">
      <datatype_1 ... />
    </ColumnLabel>
    <ColumnLabel name="title of column2">
      <datatype_2 ... />
    </ColumnLabel>
    ...
  </Table>
```

```
</Line>
```

For example:

```
<Line>
  <Table>

    <ColumnLabel name="Date">
      <DateAndTime Date="yes" Time="no" />
    </ColumnLabel>

    <ColumnLabel name="XYZ Status">
      <CheckBox name="XYZ" checked="on" />
    </ColumnLabel>

    <ColumnLabel name="Integer Value">
      <Integer />
    </ColumnLabel>

    <ColumnLabel name="Floating Value">
      <Double />
    </ColumnLabel>

    <ColumnLabel name="Who?">
      <Select editable="yes">
        <Option name="me">
        <Option name="you">
      </Select>
    </ColumnLabel>

    <ColumnLabel name="some text">
      <Field columns="30" rows="1" />
    </ColumnLabel>

  </Table>
</Line>
```

There is more information on constructing the XML code and a more detailed example in section 18.1 *Form Definition Files*.

11.5.4 Create Forms with Selected Reloadable Fields

What?

You can create a form that is a composite of its own elements and of one or more other forms. The component forms may in turn be composites of yet other forms, ad infinitum. You can configure component forms to reload

previously saved data or not. You can configure component forms such that their data get saved to a reload area each time an entry of the end form type is archived, or such that the data are not saved each time.

Why?

Why would you want to embed forms within forms? This technique allows you to collect all the information your experiment needs in the entry while minimizing data input by the user. You can create forms that are simple for the user with some fields initially filled in with new data (e.g., current date and time), other fields containing previously saved information (either a constant value or data from the previous entry of the form), and still other fields blank. All the fields remain editable.

How?

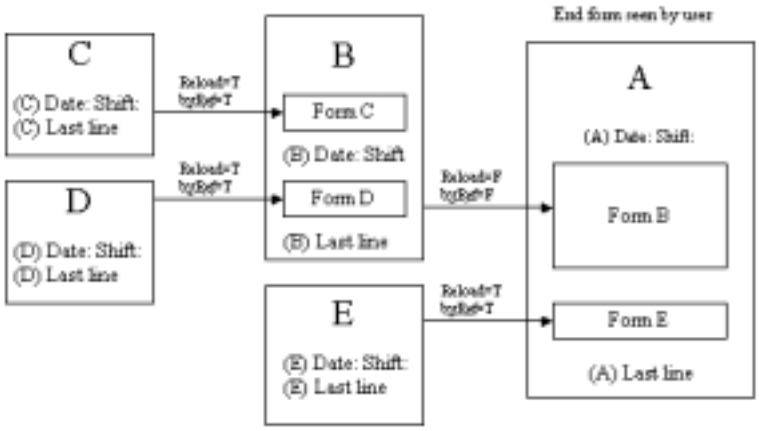
To embed one (source) form inside another (target), you create a separate XML form definition file for each, then in the target file, use the `<insertform>` element (in place of a `<Line>` element), e.g.,:

```
<insertform
  name="source_form.xml"
  reload="false"
  byReference="false"
/>
```

The name attribute is the source form definition filename. The reload and byReference attributes both take values of true or false. If reload="true", then the source form is inserted into the target with reloaded data; if "false", it's inserted blank. We'll discuss byReference further on; its function comes into play when the end form is either archived or checkpointed.

The `<insertform>` element will not work inside a repeat block.

Let's take an example, illustrated by the schematic below. The XML code and other details are given in section 18.1.8 *Sample Form with Embedded Forms*. Here we discuss the concepts.



Form A is the end form that the user sees. It is composed of some native elements (**DATE:** and **SHIFT:** line at the top, and the **LAST LINE** at the bottom) and two inserted (source) forms, B and E. E is a simple form with native elements only, whereas B (as a target) contains (source) forms C and D in addition to some native elements.

Forms C and D are both inserted into B with reload set to true. Form B is inserted into A with `reload="false"`. Form E is inserted into A with `reload="true"`. Form A is marked `reload="false"` (in its definition file). End result:

- A's native elements are either blank or filled in with new data (e.g., Date and Time). (If A were set to `reload="true"`, then user would get a prompt asking whether to reload or not.)
- B's native elements are either blank or filled in with new data (e.g., Date and Time) since reload is false. (An intermediate target form such as B should always be set to `reload="false"`.)
- The portion of B that is form C gets passed to A as it appears in B, i.e., with reloaded data. If there is no previously saved data for one or more of C's fields, those fields will be blank.
- The portion of B that is form D gets passed to A as it appears in B, i.e., with reloaded data. If there is no previously saved data for one or more of D's fields, those fields will be blank.
- The portion of A that is form E contains data from E's reload file, since reload is true. If there is no previously saved data for one or more of its fields, those fields will be blank.

Now the user edits form A. Then user archives the entry, and pulls up a new one. What will the new form entry display? In this case, (`reload="false"` for A) the portions native to A are either new or blank. The non-native portions depend upon the value of the byReference attribute set on insert:

- If `byReference="true"` for a component (or subcomponent) form of A, then when A is archived or checkpointed, the data for that component gets written to a reload file[1] in the forms directory, overwriting any previous instance of the file.
- If `byReference="false"` for a component (or subcomponent) form of A, then data from that component is never written to a reload file.

Create Field with Constant Default Value

Say you want the same default value to always appear in a particular field of your end form. You want the user to be able to change it on any given entry, but not to overwrite the reload value. To accomplish this, include that field in a (source) form that gets inserted into the (target) end form (or into one of the end form's component forms). Follow the procedure outlined here:

1) In the target form's definition file, set `reload="true"` and `byReference="true"` upon insertion of source form.

2) Invoke **CRL**, and create an entry using the end form. Either archive the entry or allow it to checkpoint, in order to create the necessary reload files in the forms directory. The field is now initialized.

3) Re-edit the target form definition file, and change byReference to false (keep reload set to true). The reload file for the source form, and hence the field's default value, will never get overwritten now.

Helpful Hints

- All the form definition files should be maintained in the forms directory as defined in the properties file.
- Typically, you want the end form's native elements not to be reloadable; use inserted forms to implement reloadable fields. To do so, set `reload="false"` in the end form's definition file.
- Any time byReference is true, then reload should also be true. Otherwise the information in the reload file will never get used.
- Within a chain of embedded forms (e.g., C inserted into B inserted into A), whenever an intermediate source form is inserted into a target form with `reload="true"` and there is a copy to reload, the source form's inserted forms are ignored; just the latest image of the intermediate source form is used. So, you should insert forms as reloadable only at the start of the chain (e.g., C inserted as reloadable into B, B inserted as NOT reloadable into A).

---

1. The filename of a reload file is the same as the corresponding form definition file with `RELOAD` prepended. For example, a reload file for the form `c.xml` would be `RELOADc.xml`.

11.6 Enabling Document View plus Thumbnail for PS and PDF File Entries

CRL gives you the option to include a PostScript or PDF document (with extension `.ps`, `.eps`, or `.pdf`) in a `LogEntryRoot` type entry, similarly to a **ROOT** file. It can be configured to create[1] and display a thumbnail image of the document in the entry. The user would click on the displayed thumbnail to view the entire document in a PS or PDF viewer, in a separate window. The thumbnail image is saved with the entry.

To enable these features, you must specify values for the following four properties in the `LogbookConfigParams.properties` file. See Chapter 15: *CRL's Java Properties* for details on each of these properties:

`Logbook.utils.imagemagick`

defines the path to ImageMagick's convert utility which converts the PS or PDF file to a `.gif` or `.jpg` thumbnail image file

`Logbook.utils.psviewer`

defines the path to a PostScript viewer which is used to display the PS file when user clicks the thumbnail

`Logbook.utils.pdfviewer`

defines the path to the Acrobat PDF viewer which is used to display the PDF file when user clicks the thumbnail

`Logbook.file_location.temp_directory`

defines the directory where the temporary files for the PostScript /PDF convert utility are stored

Two notes:

- If these parameters are left blank or left out all together, **CRL** will still save the PostScript or PDF file and operate fine, but the thumbnail feature and PS/PDF viewing will not be available.
- All temporary files created in the directory specified by the property `Logbook.file_location.temp_directory` upon creation of a thumbnail are removed upon closure of the PS or PDF viewer application.

1. It's actually a separate application that creates the thumbnail.

11.7 Editing the Inquiries Configuration File

The Logbook inquiry XML configuration file governs the fields on which you can query when using the inquiry feature described in section 8.2 *Inquiries*. The DTD and default XML files are listed in section 18.2 *The Logbook Inquiry Configuration File*.

Most **CRL** administrators will have no need to edit these files.

Only edit the XML file if you want to disable/re-enable any of the filters used for inquiries. The inquiry DTD and XML files must be kept in the directory defined in the properties file (see Chapter 15: *CRL's Java Properties*) by the parameter:

```
Logbook.file_location.inquiries_directory
```

11.8 Configuring Print Queues

There are no issues regarding printing from **CRL** installed on a Windows OS.

Fermilab flpr queues are not available; if running **CRL** on Linux, printing requires the **lp** print service to be setup. For KDE, see the Computing Division web page *KDE How-To - Using Printers* at `http://www-oss.fnal.gov/projects/fermilinux/611/adminclass/printers.full.html`.

Other Linux windowing systems provide similar interfaces. There are man pages describing how to do the configuration from the command line using the **`lpadmin`** command.

11.9 Starting the Process Logger Daemon

The Process Logger, described in Chapter 9: *Programmer's Guide to the Process Logger*, is run as a standalone daemon process that monitors specified TCP ports for input, interprets the input as **CRL** entries, and creates and logs the entries. There can be multiple back-to-back messages on a single open TCP connection and many concurrent TCP connections on any TCP port. Each experiment must assign and make known the TCP port number(s) for remote program connections. If your experiment has multiple processes that will create entries in **CRL**, you may want to consider running each on a separate TCP port. The advantage is that you can turn off one process at a time by simply restarting the daemon without that process' assigned port.

The **CRL** administrator needs to first obtain the Plog software from the **CRL** development group (contact *crl-dev@fnal.gov*). There is no Plog-related configuration required in the **CRL** configuration file. Start the Plog daemon at any time before running a program that sends messages to it. Use the following command to start it. (We recommend that you create a script that takes the port number(s) as argument(s), and runs this command.)

```
% java -jar LogBookProcessLogger.jar <CRL_admin_directory>\
  <TCP_port_number> [<additional_TCP_ports>]
```

The command arguments are defined as follows:

`LogBookProcessLogger.jar`	the executable from which Plog program is run
`<CRL_admin_directory>`	the path to the directory containing **CRL** configuration files and properties file
`<TCP_port_number>`	the TCP port number assigned to Plog
`[<additional_TCP_ports>]`	a space separated list of additional TCP port numbers assigned to Plog

The command uses the configuration information in the **CRL** admin directory to determine the **CRL** installation in which the new XML entries and their corresponding HTML web pages are to be stored. At least one port number is required on the command line.